WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Terms	Documents
L2 and L1	7

Database: US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L3

Refine Search

Recall Text | Clear

Search History

DATE: Tuesday, September 30, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=OR			
L3	L2 and L1	7	L3
L2	polygon adj4 pipeline	70	L2
L1	pipeline adj3 texture	70	L1

END OF SEARCH HISTORY

WEST

Generate Collection | Print

Search Results - Record(s) 1 through 7 of 7 returned.

☑ 1. Document ID: US 6597363 B1

L3: Entry 1 of 7 File: USPT Jul 22, 2003

DOCUMENT-IDENTIFIER: US 6597363 B1
TITLE: Graphics processor with deferred shading

Brief Summary Text (190):
Each vertex includes a color pointer, and as vertices are received, the vertices including the color pointer are stored in sort memory data storage. The color pointer is a pointer to a location in the polygon memory vertex storage that includes a color portion of the vertex data. Associated with all of the vertices, of either a strip or a fan, is an Material-Lighting-Mode (MLM) pointer set. MLM includes six main pointers plus two other pointers as described below. Each of the six main pointers comprises an address to the polygon memory state storage, which is a sequential storage of all of the state that has changed in the pipeline, for example, changes in the texture, the pixel, lighting and so forth, so that as a need arises any time in the future, one can recreate the state needed to render a vertex (or the object formed from one or more vertices) from the MLM pointer associated with the vertex, by looking up the MLM pointers and going back into the polygon memory state storage and finding the state that existed at the time.

Brief Summary Text (199):
Polygon memory vertex storage stores just the color portion. Polygon memory stores the part of pipeline stat that is not needed for hidden surface removal, and it also stores the part of the vertex data which is not needed for hidden surface removal (predominantly the items needed to make colors.)

Brief Summary Text (205):
The inventive pipeline includes a texture memory which includes a texture cache really a textured reuse register because the structure and operation are different from conventional caches. The host also includes storage for texture, which may typically be very large, but in order to render a texture, it must be loaded into the texture cache which is also referred to as texture memory. Associated with each VSP are S and T's. In order to perform trilinear MIP mapping, we necessarily blend eight (8) samples, so the inventive structure provides a set of eight content addressable (memory) caches running in parallel. n one embodiment, the cache identifier is one of the content addressable tags, and that's the reason the tag part of the cache and the data part of the cache is located are located separate from the tag or index. Conventionally, the tag and data are co-located so that a query on the tag gives the data. In the inventive structure and method, the tags and data are split up and indices are sent down the pipeline.

Brief Summary Text (211):
Well we break that conventional concept completely because the inventive structure and method are directed to a deferred shader. Even if a tile should happen to have an entire object there will typically be different background, and the inventive Cull Block and Cull procedure will typically generate and send VSPs in a completely jumbled and spatially incoherent order, even if the tile might support some degree of spatial coherency. As a result, the pipeline and texture block are advantageously capable of changing the texture map on the fly in real-time and in response to the texture required for the object primitive (e.g. triangle) received. Any requirement

to repeatedly retrieve the texture map from the host to process the particular object primitive (for example, single triangle) just received and then dispose of that texture when the next different object primitive needing a different texture map would be problematic to say the least and would preclude fast operation.

Detailed Description Text (661):
The user provides a texture number when the texture is passed from user space with OpenGL calls. The user can send some triangles to be textured with one map and then change the texture data associated with the same texture number to texture other triangles in the same frame. Our pipeline requires that all sets of texture data for a frame be available to the Texture Block. In software, we assign a virtual texture number to each texture map.

CLAIMS:

4. A graphics rendering method for forming a rendered image stored in a frame buffer, the graphics rendering method comprising the steps: receiving graphics primitives; transforming the graphics primitives; clip testing the graphics primitives to determine if the graphics primitives are at least partially contained in a view volume; face determination to discard any of the graphics primitives that are facing away from a viewing point; performing setup for incremental render for the graphics primitives, comprising the computing of derivatives; storing, in a spatial memory: (1) the part of the pipeline state needed for hidden surface removal; and (2) the part of the graphics primitives needed for hidden surface removal, comprising vertex locations and spatial derivatives; storing, in a polygon memory: (1) the part of the pipeline state not needed for hidden surface removal; and (2) the part of the graphics primitives not needed for hidden surface removal, comprising vertex colors, texture coordinates, and color derivatives; hidden surface removal to determine which portions of the graphics primitives stored in the spatial memory affect the final color of pixels in the frame buffer; fragment coloring to generate color values for each fragment, each fragment comprising a sample or a group of samples within one of the pixels, the generation of color values for each of the fragments being done only for the portions of the graphics primitives that are determined to affect the final color of pixels in the frame buffer; blending the fragment colors together to generate a single color value per pixel; and storing the generated per-pixel color values into the frame buffer.

☐ 2. Document ID: US 6525728 B2

L3: Entry 2 of 7 File: USPT Feb 25, 2003

DOCUMENT-IDENTIFIER: US 6525728 B2
TITLE: Method and apparatus for using a general three-dimensional (3D) graphics pipeline for cost effective digital image and video editing, transformation, and representation

Brief Summary Text (13):
It will be appreciated that we use a 3D graphics pipeline in a novel way. Previously, graphics pipelines were used to map a texture onto a geometric object as a method of giving the geometric object a particular appearance, and to create a new virtual environment comprising a set of such objects. This was done by providing a texture image and a geometry to a graphic pipeline to create a new virtual image. This is in contrast to using the geometry to transform and edit the images.

Detailed Description Text (7):
Instead of using primitives, such as polygons, some graphics pipelines can process geometric surface areas defined in other ways, e.g. by mathematical equations. This technique for defining geometric surface areas is called "implicit." As explained below, both techniques for defining such surface areas can be used to perform a method in accordance with our invention.

Detailed Description Text (18):
During the above-described process constructing the pixel array and providing it in the frame buffer, the pipeline a) fetches the portion of texture map 2 "tacked" to the vertices of mesh 4 (and therefore stretched over each triangle), b) determines how and where that portion of the texture map should appear, given the orientation of the triangles relative to the viewer and the location of the light source, and c) constructs the appropriate bit map pixel array for storage in the frame buffer. The contents of this frame buffer are then displayed as an image on a computer screen.

☑ 3. Document ID: US 6518974 B2

L3: Entry 3 of 7 File: USPT Feb 11, 2003

DOCUMENT-IDENTIFIER: US 6518974 B2
TITLE: Pixel engine

Detailed Description Text (30):
The Windower/Mask unit performs the scan conversion process, where the vertex and edge information is used to identify all pixels that are affected by features being rendered. It works on a per-polygon basis, and one polygon may be entering the pipeline while calculations finish on a second. It lowers its "Busy" signal after it has unloaded its input registers, and raises "Busy" after the next polygon has been loaded in. Twelve to eighteen cycles of "warm-up" occur at the beginning of new polygon processing where no valid data is output. It can be stopped by "Busy" signals that are sent to it from downstream at any time.

Detailed Description Text (81):
The Pixel Interpolator unit works on polygons received from the Windower/Mask. A sixteen-polygon delay FIFO equalizes the latency of this path with that of the Texture Pipeline and Texture Cache.

Detailed Description Text (125):
The Mapping Engine receives pixel mask and steering data per span from the Windower/Mask, gradient information for S, T, and 1/W from the Plane Converter, and state variable controls from the Command Stream Interface. It works on a per-span basis, and holds state on a per-polygon basis. One polygon may be entering the pipeline while calculations finish on a second. It lowers its "Busy" signal after it has unloaded its input registers, and raises "Busy" after the next polygon has been loaded in. It can be stopped by "Busy" signals that are sent to it from downstream at any time. FIG. 5 is a block diagram identifying the major blocks of the Mapping Engine.

Detailed Description Text (316):
The Texture Cache receives U, V, LOD, and texture state variable controls from the Texture Pipeline and texture state variable controls from the Command Stream Interface. It fetches texel data from either the FSI or from cache if it has recently been accessed. It outputs pixel texture data (RGBA) to the Color Calculator as often as one pixel per clock.

Detailed Description Text (317):
The Texture Cache works on several polygons at a time, and pipelines state variable controls associated with those polygons. It generates a "Busy" signal after it has received the next polygon after the current one it is working on, and releases this signal at the end of that polygon. It also generates a "Busy" if the read or fetch FIFOs fill up. It can be stopped by "Busy" signals that are sent to it from downstream at any time.

Detailed Description Text (328):
This look up table (LUT) is a special purpose memory that contains eight copies of 256 16-bit entries per stream. The palette data is loaded and must only be performed after a polygon flush to prevent polygons already in the pipeline from being processed with the new LUT contents. The CSI handles the synchronization of the palette loads between polygons.

Detailed Description Text (402):
The Arithmetic Stretch Blitter is implemented in the 3D pipeline using the texture mapping engine. The original image is considered a texture map and the scaled image is considered a rectangular primitive, which is rendered to the back buffer. This provides a significant gate savings at the cost of sharing resources within the device which require a context switch between commands.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 4. Document ID: US 6456287 B1

L3: Entry 4 of 7 File: USPT Sep 24, 2002

DOCUMENT-IDENTIFIER: US 6456287 B1
TITLE: Method and apparatus for 3D model creation based on 2D images

Detailed Description Text (7):
Instead of using primitives, such as polygons, some graphics pipelines can process geometric surface areas defined in other ways, e.g. by mathematical equations. This technique for defining geometric surface areas is called "implicit." As explained below, both techniques for defining such surface areas can be used to perform a method in accordance with our invention.

Detailed Description Text (18):
During the above-described process constructing the pixel array and providing it in the frame buffer, the pipeline a) fetches the portion of texture map 2 "tacked" to the vertices of mesh 4 (and therefore stretched over each triangle); b) determines how and where that portion of the texture map should appear, given the orientation of the triangles relative to the viewer and the location of the light source; and c) constructs the appropriate bit map pixel array for storage in the frame buffer. The contents of this frame buffer are then displayed as an image on a computer screen.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 5. Document ID: US 6452594 B1

L3: Entry 5 of 7 File: USPT Sep 17, 2002

DOCUMENT-IDENTIFIER: US 6452594 B1
TITLE: Method and apparatus for using a 3D graphics pipeline and 3D imaging for cost effective watermarking

Detailed Description Text (8):
Instead of using primitives, such as polygons, some graphics pipelines can process geometric surface areas defined in other ways, e.g. by mathematical equations. This technique for defining geometric surface areas is called "implicit." As explained below, both techniques for defining such surface areas can be used to perform a method in accordance with our invention.

Detailed Description Text (19):
During the above-described process of constructing the pixel array and providing it in the frame buffer, the pipeline a) fetches the portion of texture map 2 "tacked" to the vertices of mesh 4 (and therefore stretched over each triangle); b) determines how and where that portion of the texture map should appear, given the orientation of the triangles relative to the viewer and the location of the light source; and c) constructs the appropriate bit map pixel array for storage in the frame buffer. The contents of this frame buffer are then displayed as an image on a computer screen.

☑ 6. Document ID: US 6342884 B1

L3: Entry 6 of 7 File: USPT Jan 29, 2002

DOCUMENT-IDENTIFIER: US 6342884 B1
TITLE: Method and apparatus for using a general three-dimensional (3D) graphics pipeline for cost effective digital image and video editing, transformation, and representation

Brief Summary Text (13):
It will be appreciated that we use a 3D graphics pipeline in a novel way. Previously, graphics pipelines were used to map a texture onto a geometric object as a method of giving the geometric object a particular appearance, and to create a new virtual environment comprising a set of such objects. This was done by providing a texture image and a geometry to a graphics pipeline to create a new virtual image. This is in contrast to using the geometry to transform and edit the images.

Detailed Description Text (7):
Instead of using primitives, such as polygons, some graphics pipelines can process geometric surface areas defined in other ways, e.g. by mathematical equations. This technique for defining geometric surface areas is called "implicit." As explained below, both techniques for defining such surface areas can be used to perform a method in accordance with our invention.

Detailed Description Text (18):
During the above-described process of constructing the pixel array and providing it in the frame buffer, the pipeline a) fetches the portion of texture map 2 "tacked" to the vertices of mesh 4 (and therefore stretched over each triangle); b) determines how and where that portion of the texture map should appear, given the orientation of the triangles relative to the viewer and the location of the light source; and c) constructs the appropriate bit map pixel array for storage in the

frame buffer. The contents of this frame buffer are then displayed as an image on a computer screen.

☑ 7. Document ID: US 5831640 A

L3: Entry 7 of 7 File: USPT Nov 3, 1998

DOCUMENT-IDENTIFIER: US 5831640 A
TITLE: Enhanced texture map data fetching circuit and method

Detailed Description Text (13):
FIG. 2 illustrates a portion of the circuitry of the graphics subunit 109 including a texture engine 10, a polygon engine 12 and a pixel pipeline 16. The texture engine 10 receives polygon vertex data over bus 5 that corresponds to respective polygons to be rendered. The polygon vertex data includes data points for each vertex of the polygon. With respect to triangle polygons, each of the three vertexes contains: its own position coordinate values (x, y, z); its own color values (red, green, blue); its own texture map coordinate values (u, v), its own perspective value (w), and other required values including an identification of the texture map data for the polygon, if any. The texture engine 10 is responsible for retrieving the texture map data for the polygon and mapping the texels of the texture data onto the pixels of the polygon. Once the texture engine 10 is given the texture map coordinates (u,v) for each vertex of a triangle, it can go to the texture cache controller 250 and access the matching texels for placement into the triangle. During this process, the texture engine 10 maintains the three dimensional perspective of the surface of the polygon. Texture map data retrieval (TDA) circuit 200 performs texture map data retrieval processes in accordance with the present invention. Aside from the texture retrieval system of texture engine 10, a number of well known procedures and circuits can be used to maintain the perspective and perform the texture mapping operations implemented within texture engine 10. Texture map pixel data is supplied from the texture engine 10 to the pixel pipeline 16 over bus 14a.

Detailed Description Text (14):
The polygon engine 12 of FIG. 2 receives the polygon data over bus 5 and performs well known polygon rendering functions with respect to the position, color, and perspective of the polygon primitive. Essentially, polygon engine uses interpolation to compute the pixel positions and colors of the pixels within the polygon primitive based on the polygon vertex data. Pixel information resulting from the polygon engine 12 is forwarded to the pixel pipeline over bus 14b. The pixel pipeline 16 blends the texture data (texels) from the texture engine 10 with the pixel data from the polygon engine 12 to form a composite polygon image. The data (pixels) of the composite image are forwarded in raster encoded format over bus 18, stored in a raster encoded frame buffer (situated within graphics subunit 109, but not shown in FIG. 2), and eventually displayed on display screen 105 (FIG. 1). The above operation is performed individually for each received polygon primitive. The pixel pipeline 16, in one embodiment, contains a latency of approximately 5-7 clock cycles, depending on programmable image features.

Terms	Documents
L2 and L1	7

Display Format: KWIC Change Format

Previous Page Next Page